UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES

Sistemas de Computación

Trabajo Práctico 1

Rendimiento

Alumnos:	Carrera:	Matricula:
Angeloff, Jorge Enrique	IElec	39206918
Colque Ventura, Santiago Agustin	IComp	41041554
Pary, Joaquin Andres	IComp	41786697

Objetivo

El objetivo de esta tarea es poner en práctica los conocimientos sobre performance y rendimiento de los computadores. El trabajo consta de dos partes, la primera es utilizar benchmarks de terceros para tomar decisiones de hardware y la segunda consiste en utilizar herramientas para medir la performance de nuestro código.

Consigna

Responder a las siguientes preguntas y mostrar con capturas de pantalla la realización del tutorial descrito en time profiling adjuntando las conclusiones sobre el uso del tiempo de las funciones.

1) Armar una lista de benchmarks, ¿cuales les serían más útiles a cada uno? y ¿Cuáles podrían llegar a medir mejor las tareas que ustedes realizan a diario?

   Pensar en las tareas que cada uno realiza a diario y escribir en una tabla de dos entradas las tareas y que benchmark la representa mejor.

2) ¿Cuál es el rendimiento de estos procesadores para compilar el kernel de linux?

   Intel Core i5-13600K

   AMD Ryzen 9 5900X 12-Core

   ¿Cuál es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core?

3) Conseguir un esp32 o cualquier procesador al que se le pueda cambiar la frecuencia. Ejecutar un código que demore alrededor de 10 segundos. Puede ser un bucle for con sumas de enteros por un lado y otro con suma de floats por otro lado. ¿Qué sucede con el tiempo del programa al duplicar (variar) la frecuencia?

4) Usar la herramienta gprof para hacer un time profiling

Desarrollo

1) Los tipos de benchmarks más populares son:

- Sintéticos: Miden el rendimiento de un componente individual en un equipo.
- Aplicaciones: Miden el rendimiento global de un equipo.
- Bajo nivel: Miden el rendimiento de los componentes.
- Alto nivel: Analizan el rendimiento de la combinación de componentes/controladores/SO de un aspecto específico del sistema.

Alumnos	Tarea diaria	Benchmark
Joaquin Pary	Gaming, streaming, monitorizar paquetes de red.	- CyberPunk2077 (Gaming) - Youtube/Netflix (Streaming) - Wireshark (Analisis de paquetes de red)
Santiago Colque	Gaming, programar.	- 3DMark (Gaming) - gprof, linux perf (Programación)
Jorge Angeloff	Simulación de circuitos electrónicos. Gaming.	- Tiempo de ejecución de las simulaciones en Proteus. - GTA V

2) **¿Cuál es el rendimiento de estos procesadores para compilar el kernel de linux?**

   Intel Core i5-13600K

   AMD Ryzen 9 5900X 12-Core

   ¿Cuál es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core?

Si analizamos los benchmarks obtenidos al compilar el kernel de Linux disponibles en https://openbenchmarking.org/, podemos ver que el I5 es superior al Ryzen 9 5900X de 12 núcleos, sin embargo se ve derrotado ante la nueva versión del procesador de AMD.

Estos resultados pueden verse justificados si tenemos en cuenta las especificaciones de los 3 procesadores.

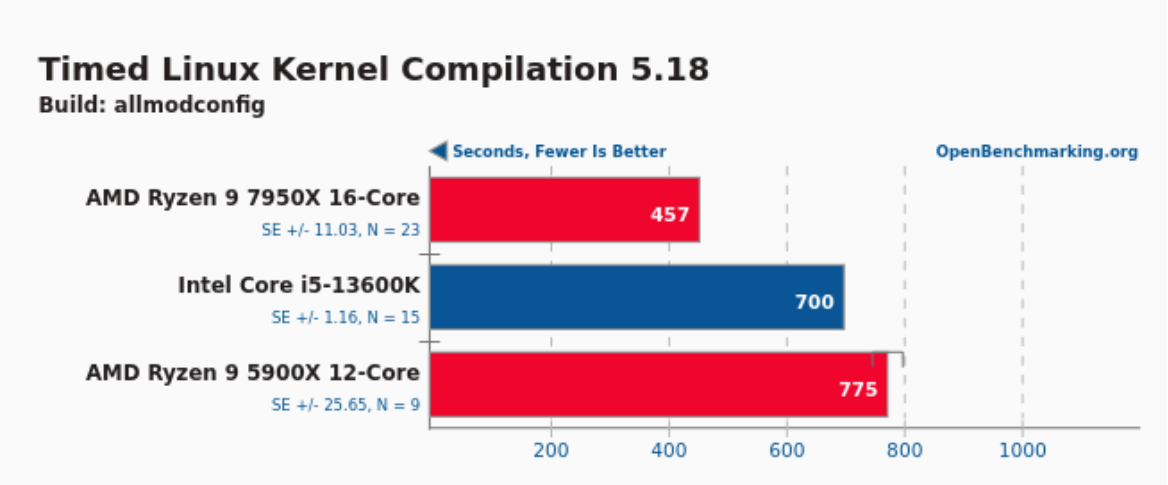
	INTEL CORE I5-13600K	AMD RYZEN 9 5900X 12-CORE	AMD RYZEN 9 7950X 16-CORE
Core Count	14	12	16
Thread Count	20	24	32
CPU Clock *	5.1 GHz	3.7 GHz	4.5 GHz
Core Family	Raptor Lake	Zen 3	Zen 4
First Appeared	2022	2020	2022
Overall Percentile	68th	67th	77th

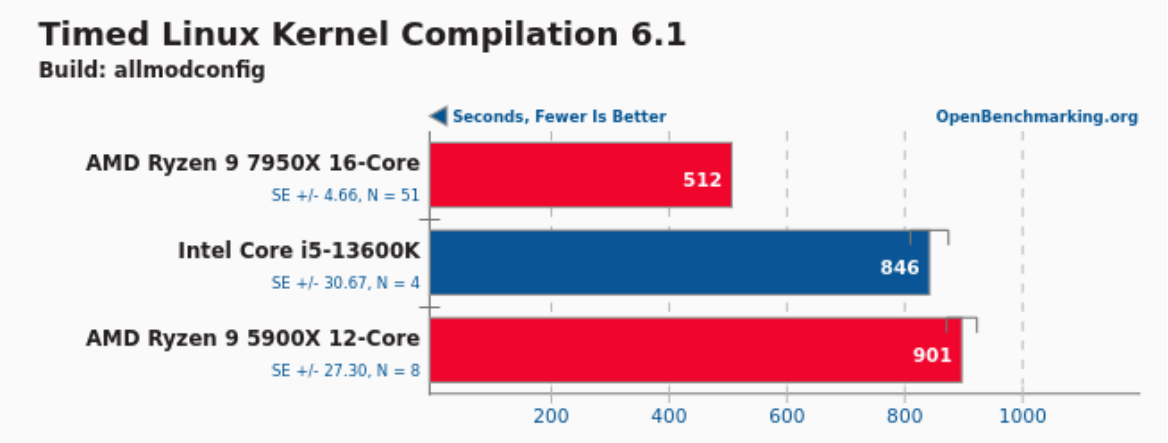
Aunque el I5 13600K tiene 4 hilos menos que el R9 5900X, es más rápido al compilar el kernel de linux, esto se explica gracias a que tiene 2 núcleos físicos extra y funciona a una frecuencia de reloj muy superior al R9.

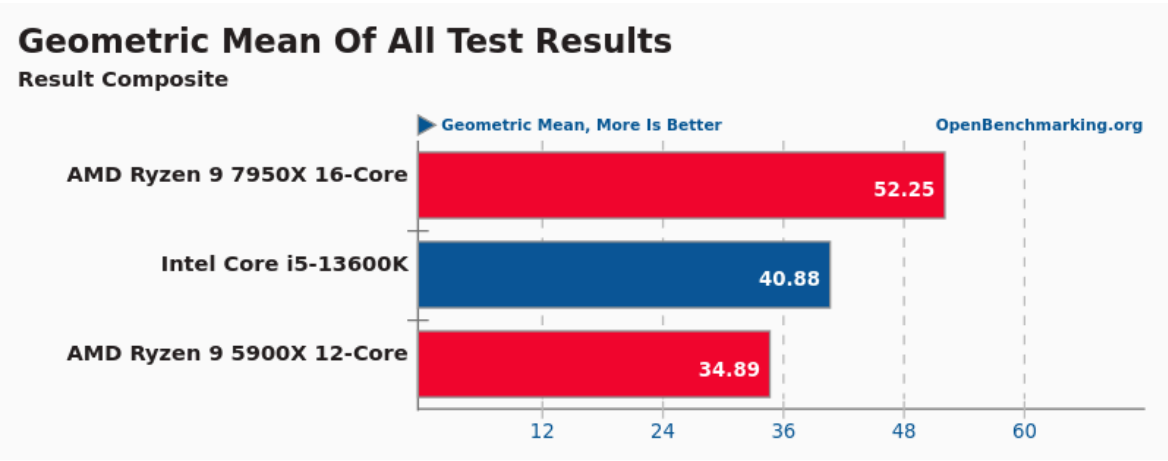
A la vez, el 7950X, posee 2 núcleos más que el I5 (también 12 hilos más), y corre a una frecuencia mayor que su antecesor, lo que le da una ventaja contra el i5.

Se pueden apreciar en la imagen de la página siguiente los siguientes resultados:

- El I5 13600K es un 10% más rápido que el R9 5900X al compilar la versión 5.18 . Esta ventaja se reduce a un 6% en la versión 6.1
- El R9 7950X es mucho más rápido. Un 41% más veloz que el R9 anterior en la versión 5.18, y un 43% en la 6.1.
- El R9 7950X es mucho más rápido que el I5. Un 35% más veloz en la versión 5.18, y un 40% en la 6.1.





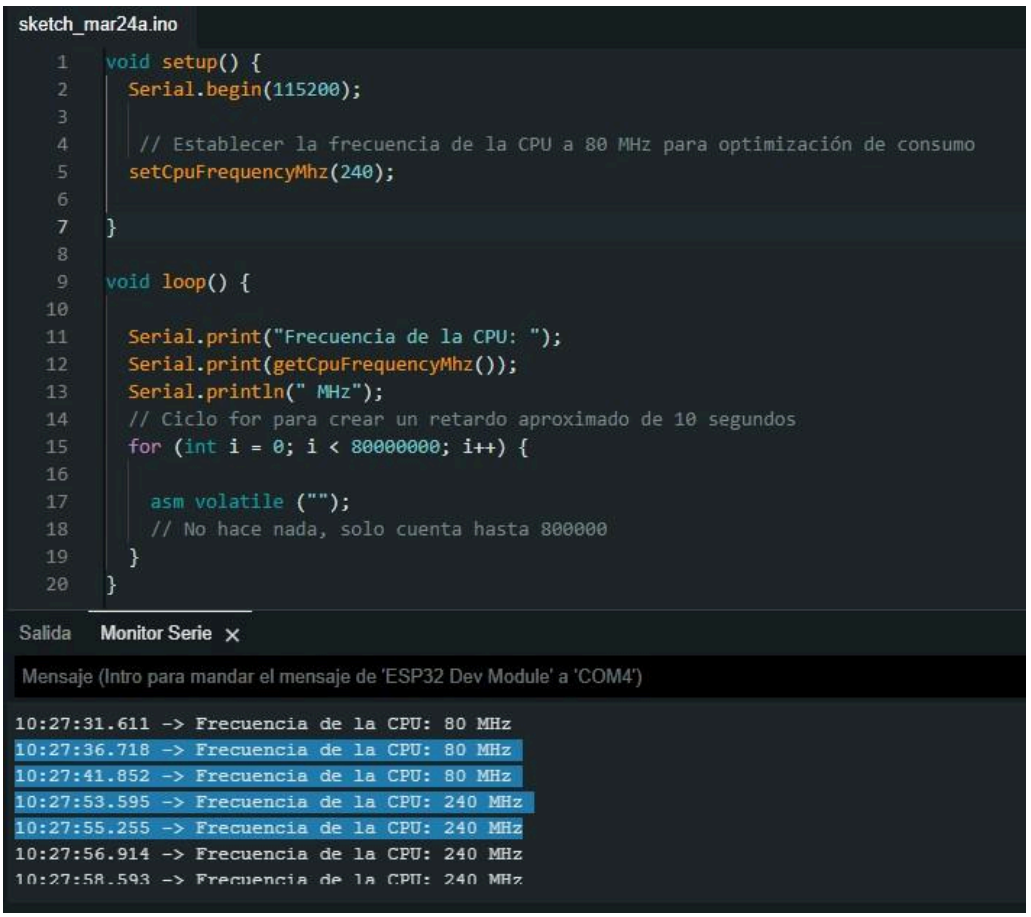


Podemos calcular el speedup o aceleración al mejorar de un R9 5900X a un R9 7950X, por lo que utilizando la versión 6.1 del kernel como referencia tenemos:

$$Speedup = \frac{EX\ CPU\ original}{EX\ CPU\ mejorado} = \frac{901}{512} = 1.75$$

Lo que indica una mejora muy grande en el rendimiento del nuevo procesador de AMD. Esta superioridad también es visible si vemos la gráfica que compara la media geométrica de todos los tests muestreados en https://openbenchmarking.org/.

3) **Resultados obtenidos al utilizar una ESP 32, con un mismo programa pero variando la frecuencia.**



Para una frecuencia de 80MHz el ciclo for tarda:

$$T_{80MHz} = 5.134\ s$$

Es decir un rendimiento:

$$\eta_{80MHz} = T^{-1}_{80MHz} = 0.195\ [s^{-1}]$$

Mientras que para el mismo ciclo pero con el CPU con una frecuencia de 240 MHz:

$$t_{240MHz} = 1.66\ s$$

Y su rendimiento es:

$$\eta_{240MHz} = T^{-1}_{240MHz} = 0.604\ [s^{-1}]$$

Puede verse que al triplicar la frecuencia el tiempo que tarda en ejecutar la misma cantidad de instrucciones se reduce 3 veces. Guardando una relación lineal entre ambas variables.

Teniendo un speedup de:

$$speedup = \frac{\eta_{240MHz}}{\eta_{80MHz}} = 3.077$$

$$eficiencia(ESP32) = \frac{speedup_n}{n} = \frac{3.077}{2} = 1.53$$

4) Time Profiling

Para realizar el time profiling utilizamos gprof. Cuando compilamos con la opción -pg, gcc inserta código adicional en el programa generado, lo que permite recopilar información durante la ejecución. Una vez que se ejecuta el programa compilado, se genera un archivo que puede ser analizado con gprof para producir un informe detallado del rendimiento de cada función, los llamados a funciones, etc.

```
santi@santi-ROG ~/Documents/SisCom2024/lab1
$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
santi@santi-ROG ~/Documents/SisCom2024/lab1
$ ./test_gprof

Inside main()

Inside func1

Inside new_func1()

Inside func2
santi@santi-ROG ~/Documents/SisCom2024/lab1
$ gprof test_gprof gmon.out > analysis.txt
santi@santi-ROG ~/Documents/SisCom2024/lab1
$ cat analysis.txt | gprof2dot | dot -Tpng -o graph.png
```

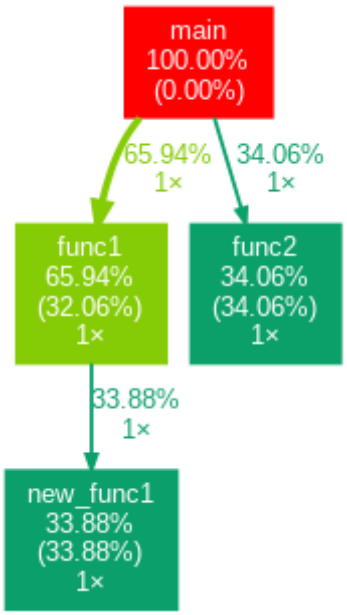
Al correr nuestro programa, se genera el archivo gmon.out, que luego utilizamos para ver los detalles con gprof.

El programa gprof genera la siguiente salida, que permite ver el tiempo ocupado en cada función, cuántas veces se ejecutó cada una, desde donde fueron llamadas, etc.

```
santi@santi-ROG ~/Documents/SisCom2024/lab1
$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 34.06      1.87     1.87        1     1.87     1.87  func2
 33.88      3.73     1.86        1     1.86     1.86  new_func1
 32.06      5.49     1.76        1     1.76     3.62  func1
```

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.18% of 5.49 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    5.49                 main [1]
                1.76    1.86       1/1           func1 [2]
                1.87    0.00       1/1           func2 [3]
-----------------------------------------------
                1.76    1.86       1/1           main [1]
[2]     65.9    1.76    1.86       1         func1 [2]
                1.86    0.00       1/1           new_func1 [4]
-----------------------------------------------
                1.87    0.00       1/1           main [1]
[3]     34.1    1.87    0.00       1         func2 [3]
-----------------------------------------------
                1.86    0.00       1/1           func1 [2]
[4]     33.9    1.86    0.00       1         new_func1 [4]
-----------------------------------------------
```

Y podemos plasmar los resultados en un gráfico, que muestra en una estructura la ejecución del programa, detallando cómo se dividió el tiempo de ejecución entre las distintas funciones y que funciones fueron llamadas desde otras.



```
santi@santi-ROG ~/Documents/SisCom2024/lab1
$ sudo perf record -g ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 7 times to write data ]
[ perf record: Captured and wrote 1,761 MB perf.data (21996 samples) ]
```

sudo perf report

```
Samples: 21K of event 'cycles:P', Event count (approx.): 24262098032
  Children      Self  Command     Shared Object         Symbol
+   99,99%     0,00%  test_gprof  libc.so.6             [.] __libc_start_call_main
+   99,99%     0,10%  test_gprof  test_gprof           [.] main
+   66,10%    32,20%  test_gprof  test_gprof           [.] func1
+   33,90%    33,90%  test_gprof  test_gprof           [.] new_func1
+   33,78%    33,78%  test_gprof  test_gprof           [.] func2
```

Como podemos observar, al analizar con linux perf, los tiempos prácticamente no varían, están distribuidos de la misma forma que con gprof y los resultados son muy parecidos.

Al ser linux perf menos invasivo y liviano que gprof, ya que no se inyecta código si no que se muestrea la ejecución, se convierte en una buena alternativa si las mediciones a realizar no necesitan tanta precisión.